Stack

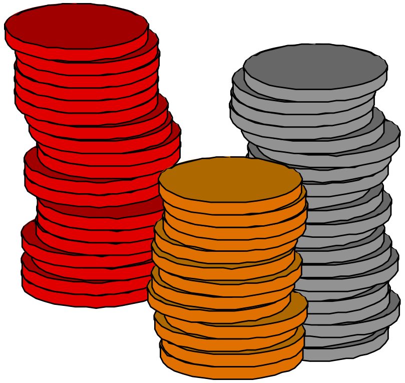
Bruce Nan

Stack

- Stack: what is a stack?
- Implementation
- Applications

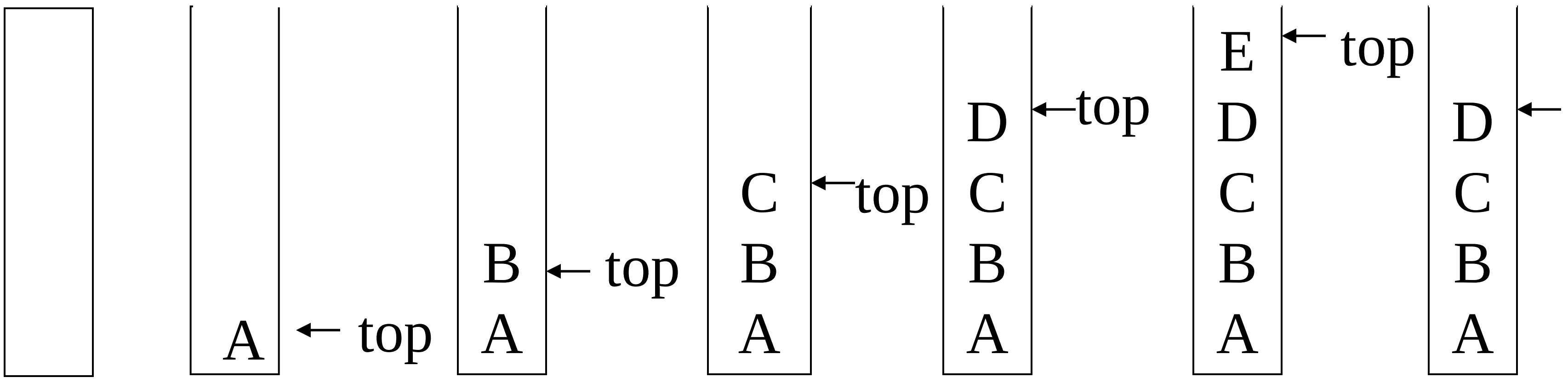
What is a stack?

- Stores a set of elements in a particular order
- Stack principle: LAST IN FIRST OUT
- = LIFO
- It means: the last element inserted is the first one to be removed
- Example



- Which is the first element to pick up?

Last In First Out



Stack Applications

- Real life
  - Pile of books
  - Plate trays
- More applications related to computer science
  - Program execution stack (read more from your text)
  - Evaluating expressions

Stack Abstract Data Type

objects: *a finite ordered list with zero or more elements.*
methods:

```
 Stack push(stack, item) ::=
          if (IsFull(stack)) stack_full
          else insert item into top of stack and return
Boolean isEmpty(stack) ::=
                     if(top ==-1)  return TRUE
                    else return FALSE
Element pop(stack) ::=
                     if(IsEmpty(stack)) return
                    else remove and return the item on the top
                         of the stack.
```

Array-based Stack Implementation

- Allocate an array of some size (pre-defined)
  - Maximum N elements in stack
- Bottom stack element stored at element 0
- last index in the array is the *top*
- Increment *top* when one element is pushed, decrement after pop

Stack Implementation: isEmpty

```
element stack[MAX_STACK_SIZE];
int top = -1;

Boolean isEmpty(Stack) ::= top< 0;
```

Push

```
void push(int *top, element item)
{
    stack[++*top] = item;
}
```

Pop

```
element pop(int *top)
{
 /* return the top element from the stack */
    if (*top == -1)
        return stack_empty( );  /* returns and error key */
    return stack[(*top)--];
 }
```

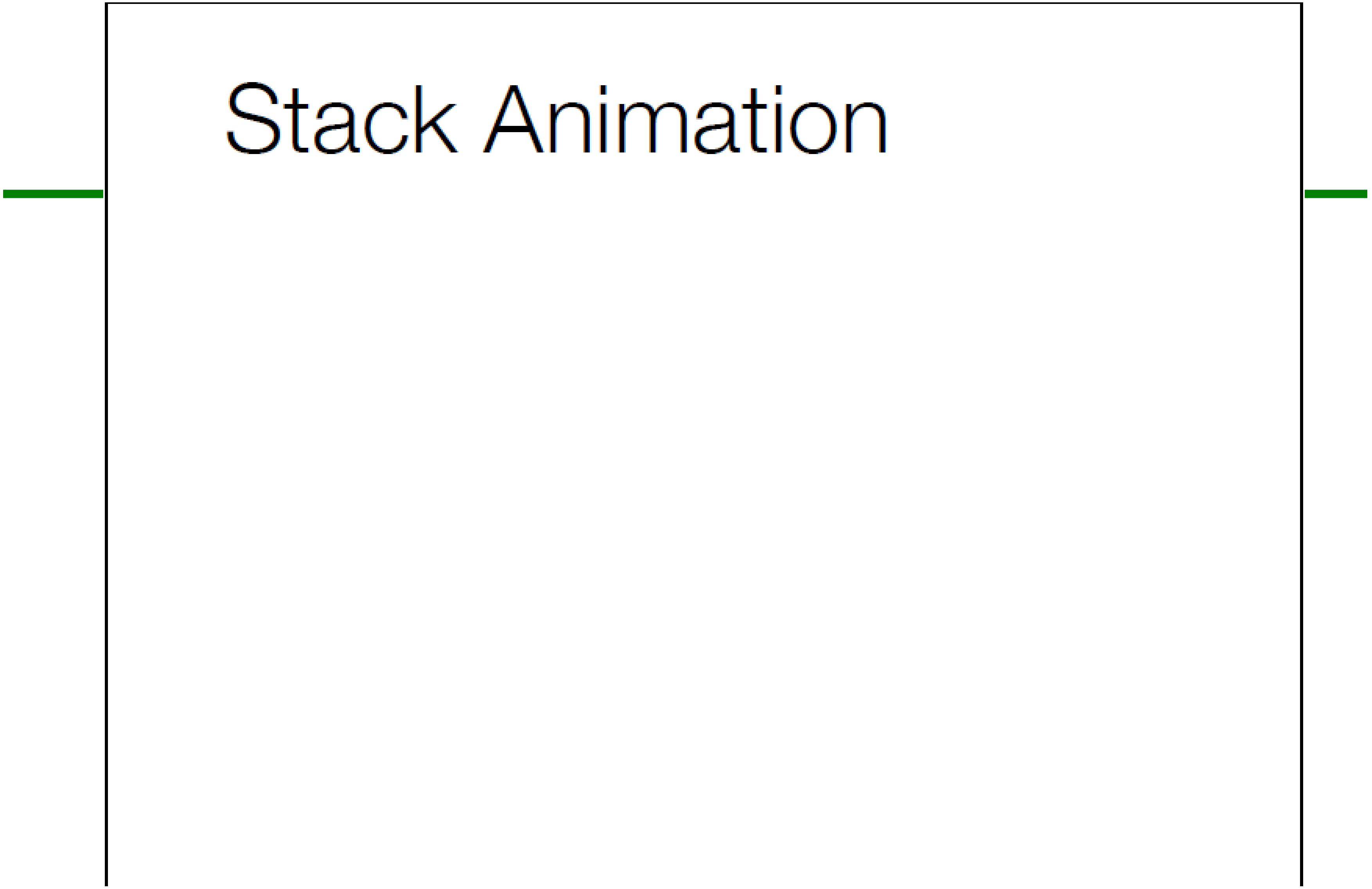
Stack Application

- Recursion
- Parsing text: infix vs. postfix
- Syntax checking ( ), { }, “”

Evaluating Recursion

- Push recursive calls onto a Stack, evaluate top
- Consider computing factorials:
  - N! = N * (N-1)!
  - 1! = 1

Stack Animation



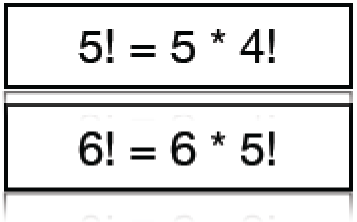
Stack Animation

6!

Stack Animation

6! = 6 * 5!

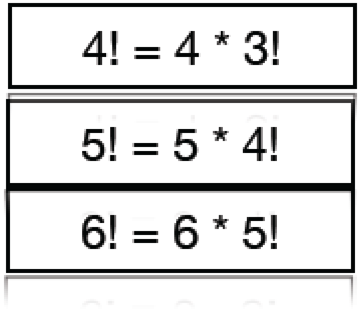
Stack Animation



5! = 5 * 4!

6! = 6 * 5!

Stack Animation

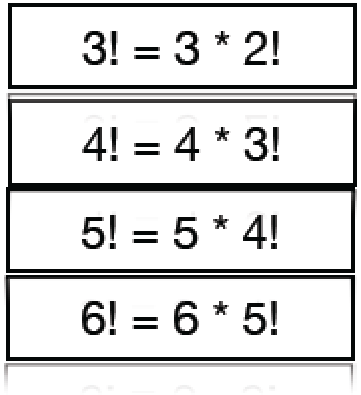


4! = 4 * 3!

5! = 5 * 4!

6! = 6 * 5!

Stack Animation



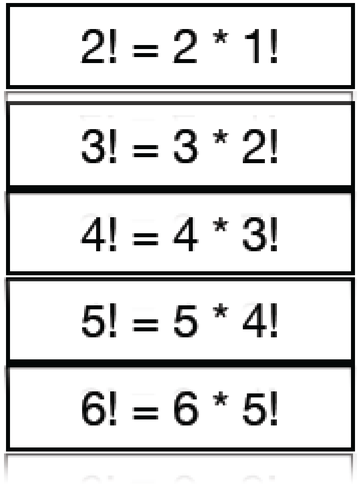
3! = 3 * 2!

4! = 4 * 3!

5! = 5 * 4!

6! = 6 * 5!

Stack Animation



2! = 2 * 1!

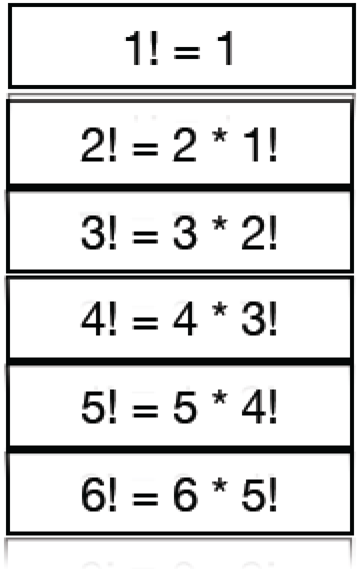
3! = 3 * 2!

4! = 4 * 3!

5! = 5 * 4!

6! = 6 * 5!

Stack Animation



1! = 1

2! = 2 * 1!

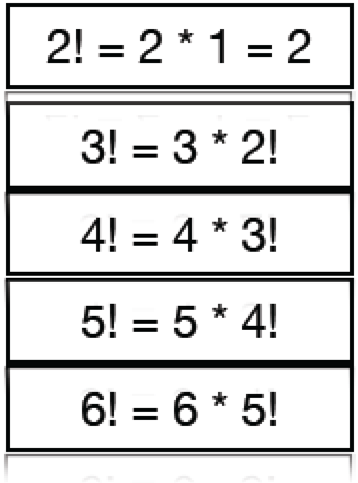
3! = 3 * 2!

4! = 4 * 3!

5! = 5 * 4!

6! = 6 * 5!

Stack Animation



2! = 2 * 1 = 2

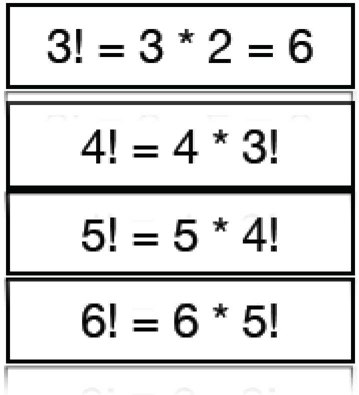
3! = 3 * 2!

4! = 4 * 3!

5! = 5 * 4!

6! = 6 * 5!

Stack Animation



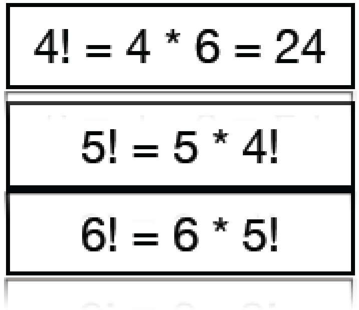
3! = 3 * 2 = 6

4! = 4 * 3!

5! = 5 * 4!

6! = 6 * 5!

Stack Animation

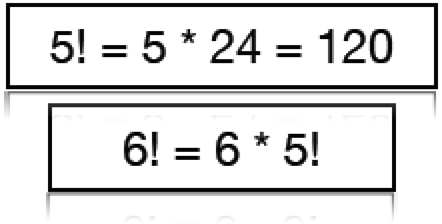


4! = 4 * 6 = 24

5! = 5 * 4!

6! = 6 * 5!

Stack Animation



5! = 5 * 24 = 120

6! = 6 * 5!

Stack Animation

6! = 6 * 120 = 720

C++ Stack

```
#include <stack>

stack<int>  st;
```

empty()	Test whether stack is empty;
size()	Return stack size;
top()	Access top element;
push()	Add element;
pop()	Remove element;

Java Stack

Stack()	Creates an Stack;
empty()	Test if empty;
peek()	Looks at the object at the top of this stack without removing it from the stack;
pop()	Removes the object at the top of this stack and returns that object;
push()	Pushes an item onto the top of this stack;
search()	Returns the 1-based position where an object is on this stack.